Calling RFC / BAPI from an EJB and Expose EJB as a Web Service

Applies to:

EJB 2.0 +, SAP Portal 7.0+. For more information, visit the Web Services homepage.

Summary

The article describes the concept and procedure of Calling RFC / BAPI in J2EE EJB application and exposing EJB as a Web Service.

Author: Nilesh Rane

Company: Larsen & Toubro Infotech Limited

Created on: 21 March 2010

Author Bio

Nilesh Rane is a SAP NetWeaver Consultant in Larsen & Toubro Infotech Limited. He has three years of SAP Portal experience. He has worked extensively in WebDynpro - Java, EPortal, J2EE, SOAP Web Services, Adobe flex & Java Dictionary

Table of Contents

Concepts

Purpose

EJB applications provide a variety of advantages and functions that make them suitable when you develop large, scalable, and flexible applications that will be used by multiple users simultaneously. EJB applications usually contain the whole business logic and therefore client developers can concentrate on the presentation logic and create thin clients.

You can develop an EJB application if:

1. You want to develop a transactional and secure application
2. Your application will be accessed by multiple clients.
3. The EJB architecture allows concurrent access to shared resources.
4. Your application will be accessed by a variety of clients.
5. EJB applications are integrated with different technologies, which enable different clients to access one and the same EJB application without needing to change the source code in the EJB application.

Since EJB applications are platform-independent, you can reuse your EJB components without having to change and recompile your source code[1].

Web services are modules in modern, service-oriented software architectures. With the help of Web services, IT infrastructures can be converted – step by step – into service-oriented architectures.

A Web service is a modularized, executable unit. It can be called in heterogeneous system landscapes and is not restricted to a single host system. Each output is determined based on the given input parameters. This output is then passed back to the caller. For example, Web services can serve to perform a credit card check, convert a currency amount, send a price query to a provider, or simply to place an order. Also, industrial manufacturers can provide their customers, partners, and suppliers with Web services for implementation in their own programs in order to set up cross-system supply chain solutions[2]

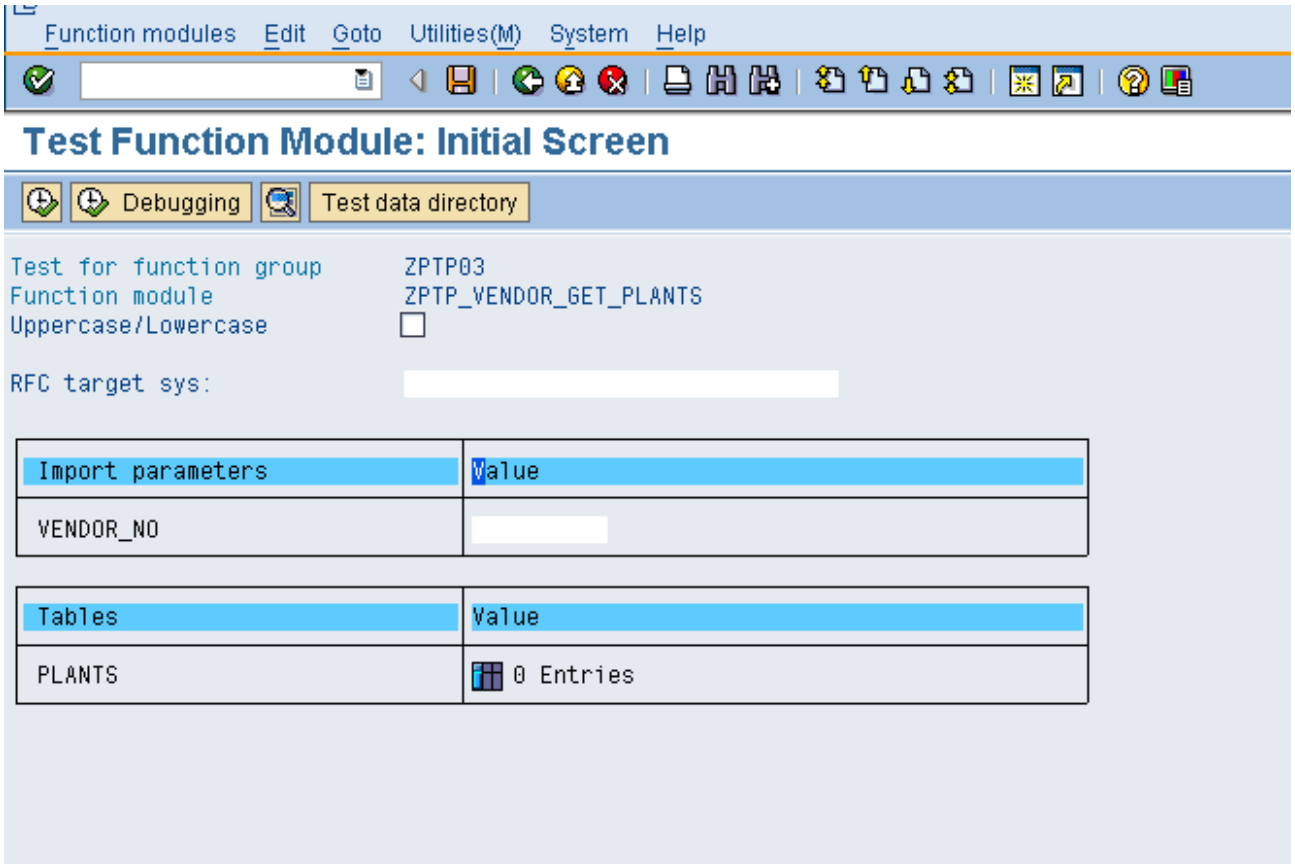
Calling RFC function from EJB

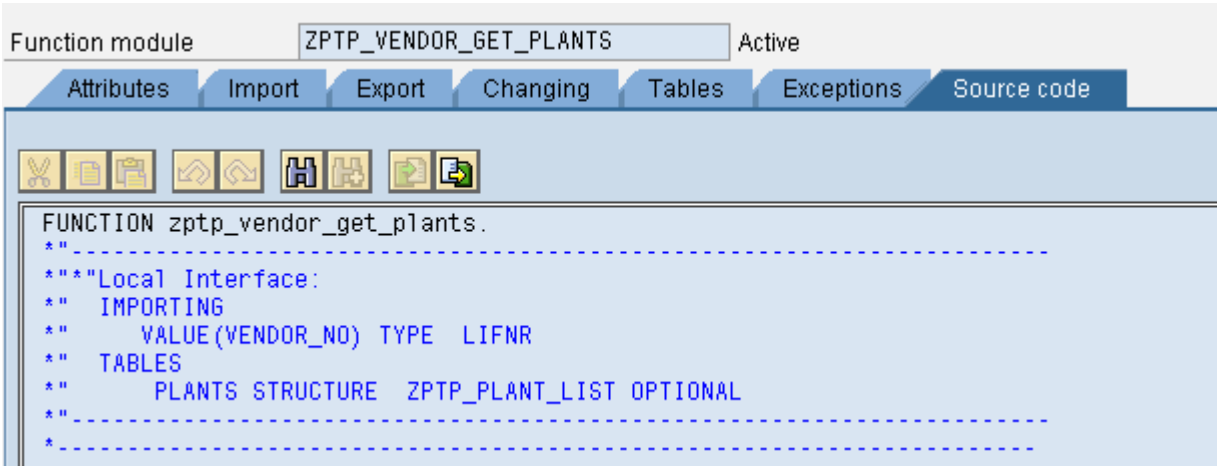
WebDynpro – Java Framework allows you to create and execute RFCs and BAPIs using JCOs. If you want to execute an RFC / BAPI from Java Class or an EJB or Portal Component, you need to create JCO dynamically or using JCO API, access JCO which is already created on WEB AS and create a connection to the SAP system to retrieve the data.

RFC Function Module

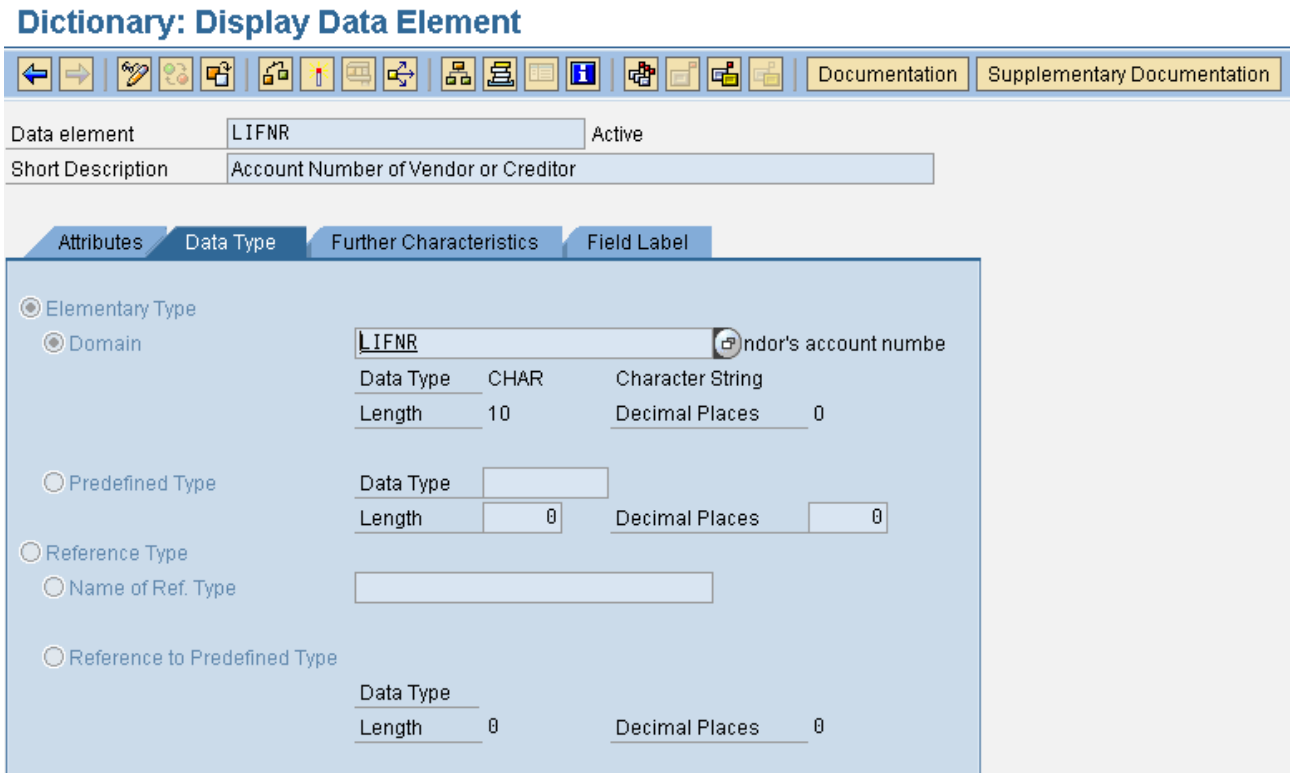
RFC ZPTP_GET_VENDOR_PLANTS is created in the SAP system which will accept vendor code as an input parameter and will produce Plants associated to it as output.

1. Interface

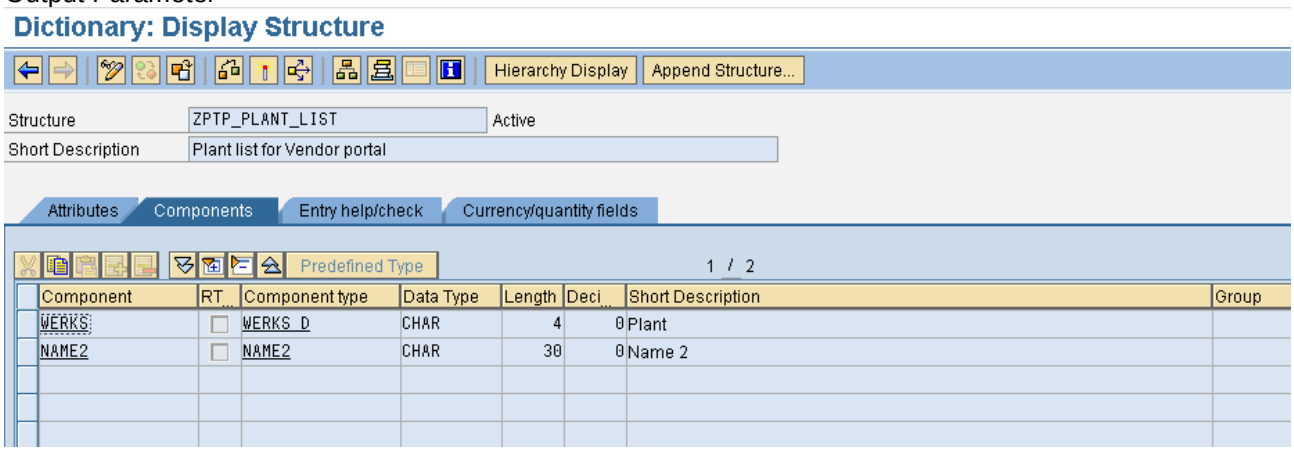




2. Input Parameter



3. Output Parameter

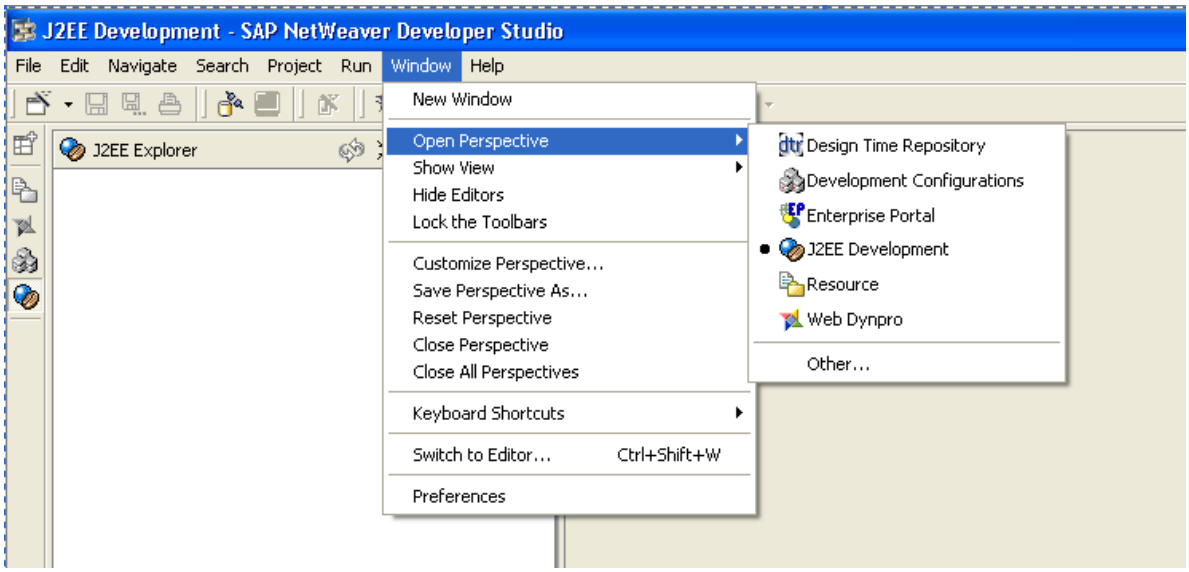


Java Connector

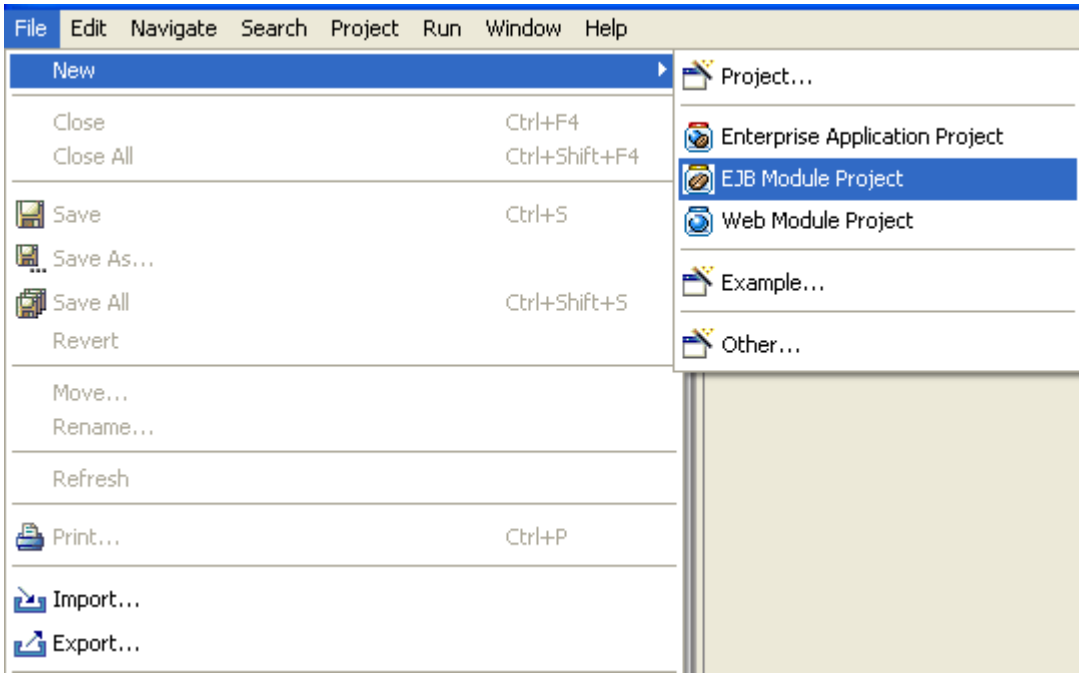
Define a JCO (`WD_MODELDATA_DEST`) on your WEB AS which will connect to the required backend SAP System.

Developing an EJB Module Project

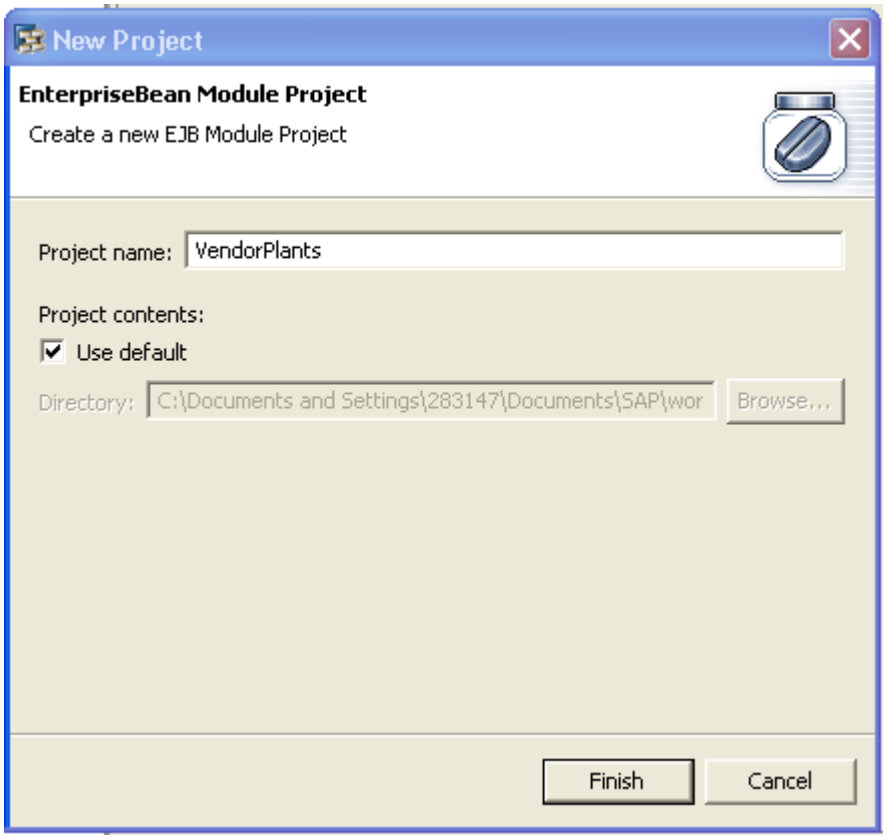
- Open NWDS in J2EE Development Perspective



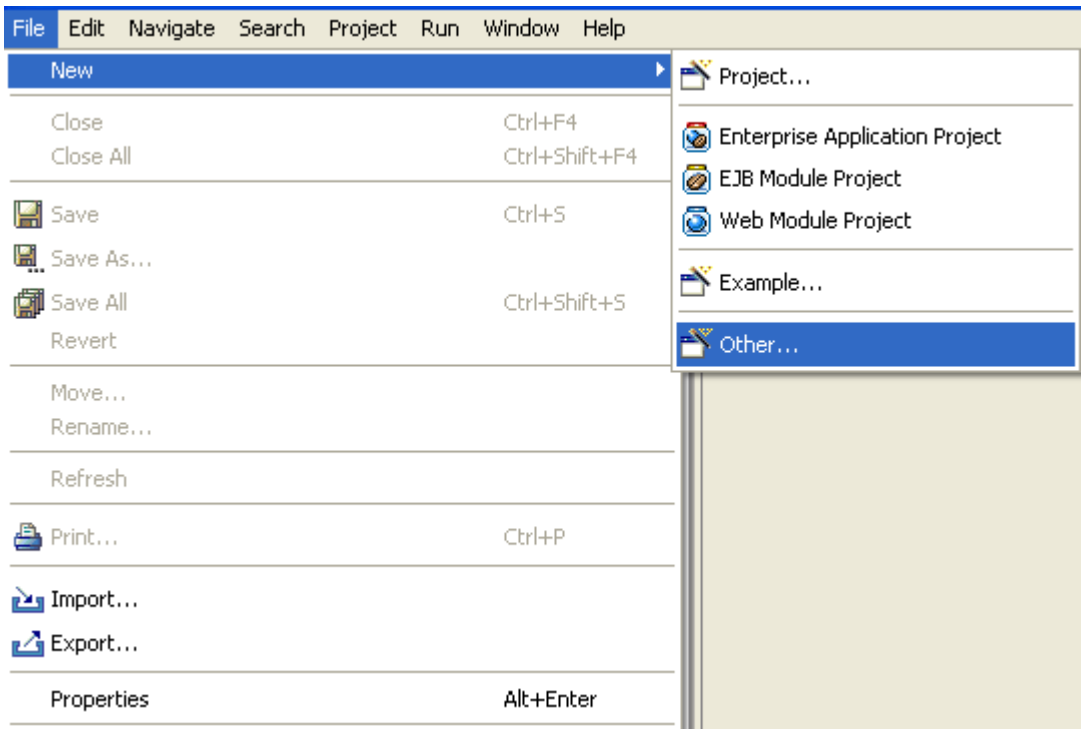
- Select File -> New -> EJB Module Project



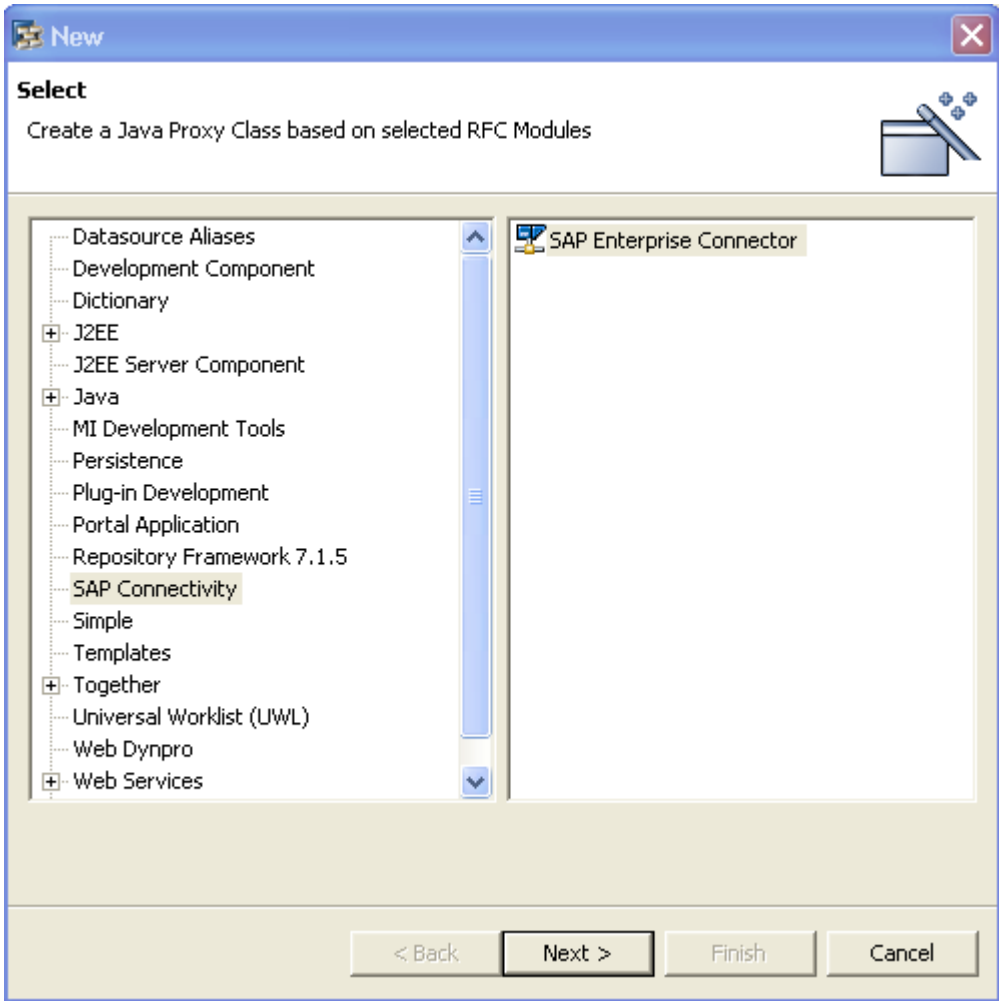
- Give any meaningful name and click on finish.



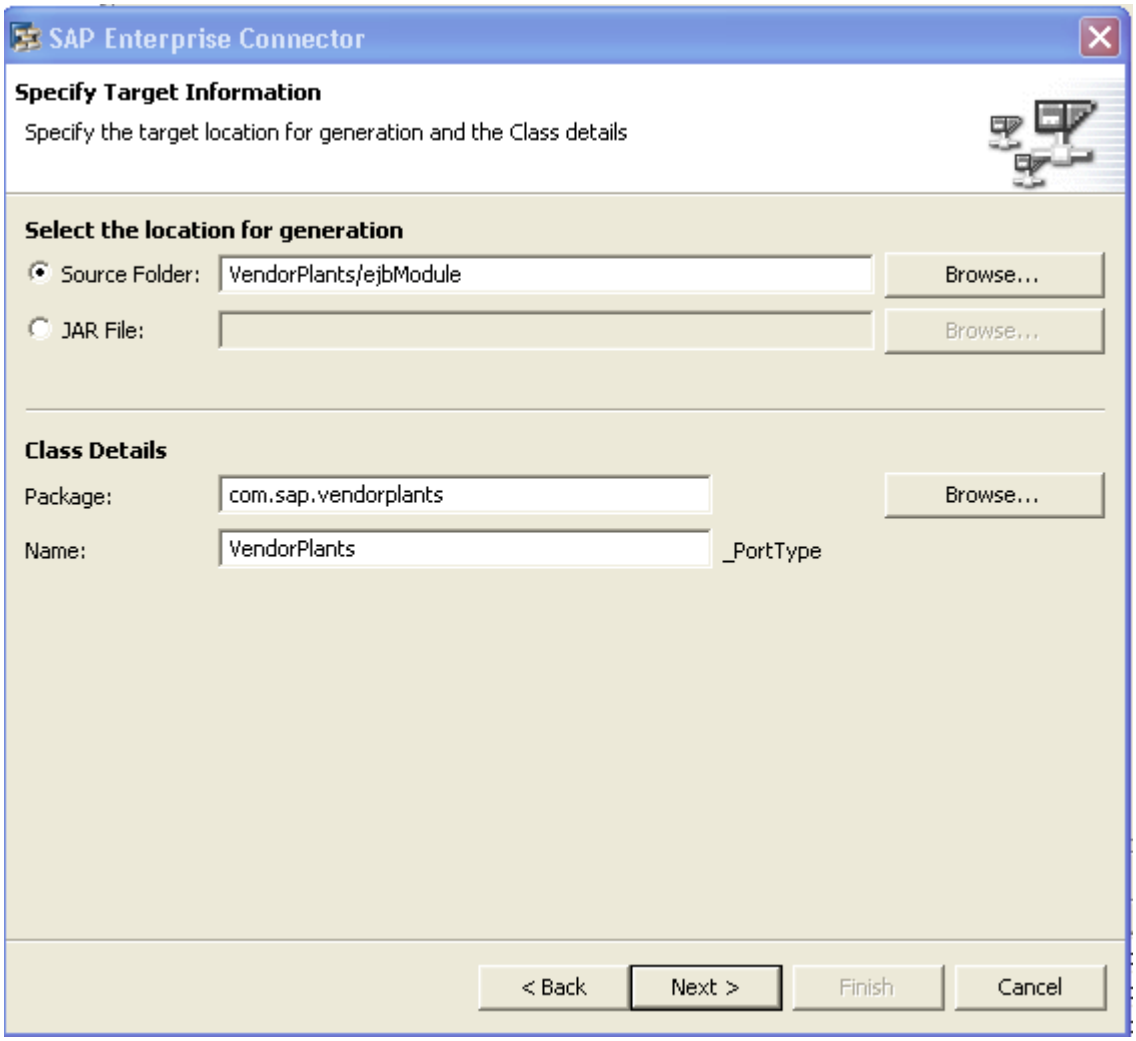
- You need to import the RFC Function Module classes to your EJB Module Project, for that Select your project and select File -> New -> Other



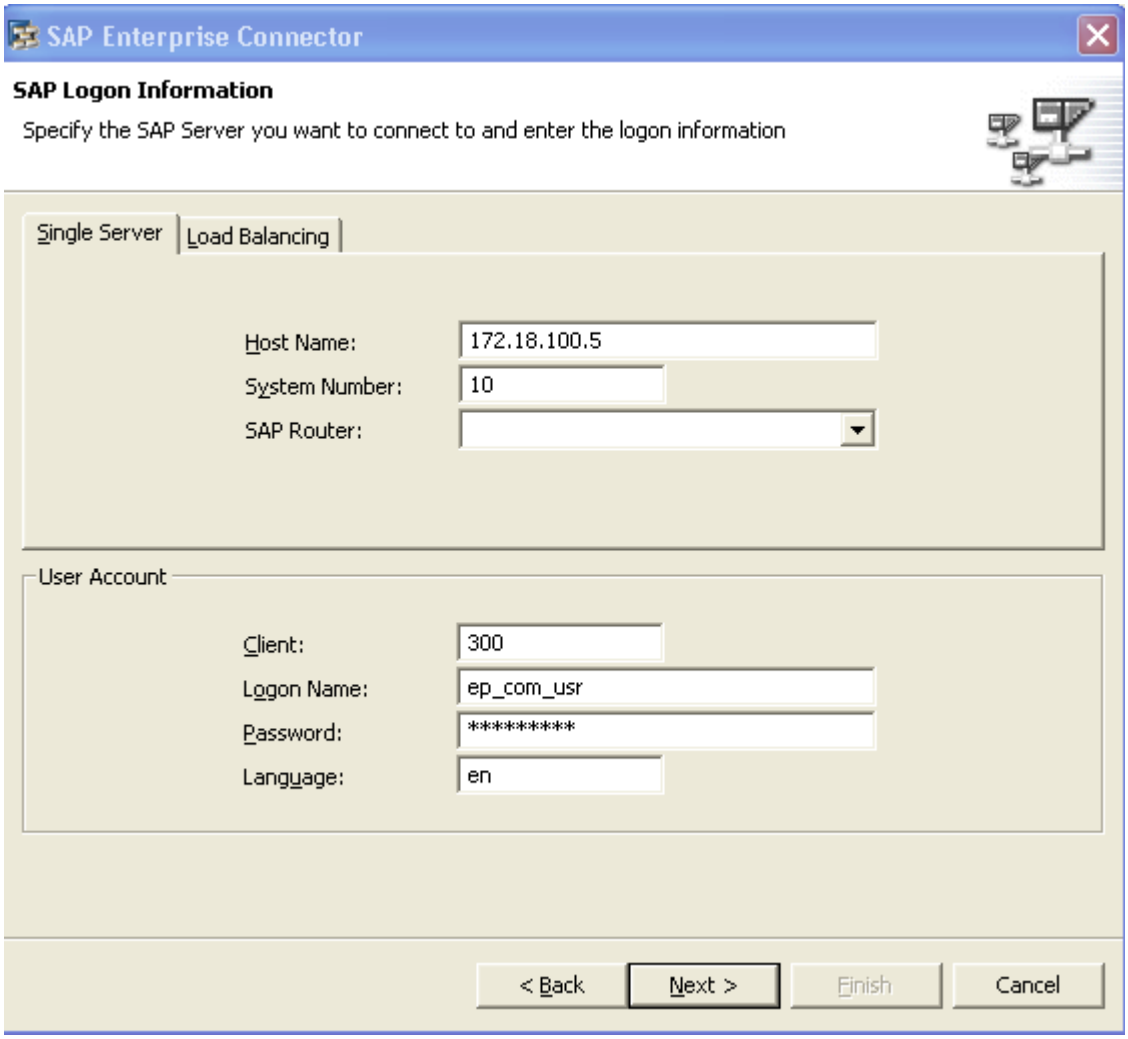
- Select SAP Connectivity and SAP Enterprise Connector. Click on next button.



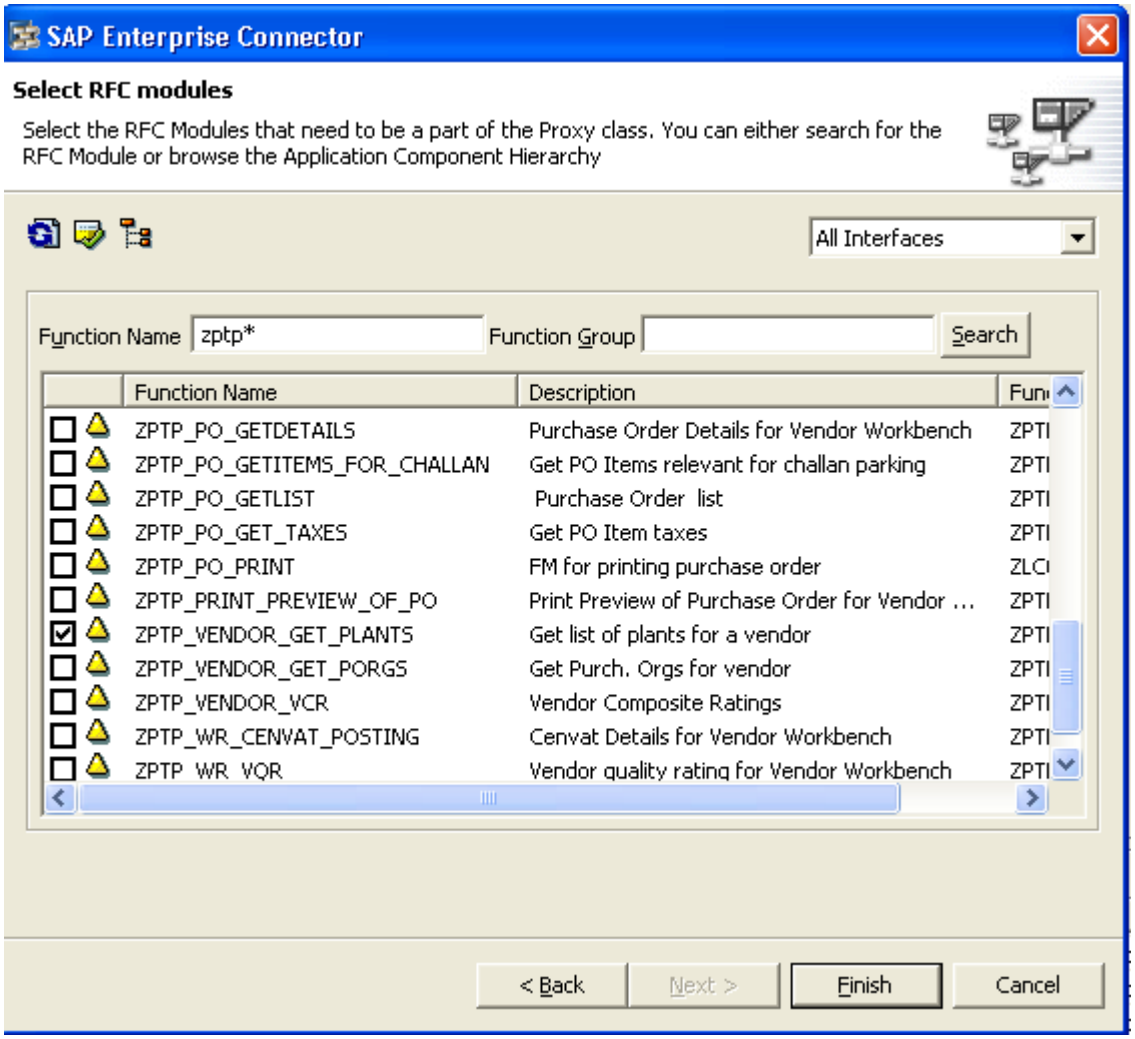
- Enter package name and Proxy Port and click on Next Button.



- Provide SAP system details and click on Next button.



- Select the required Function Module and click on Finish

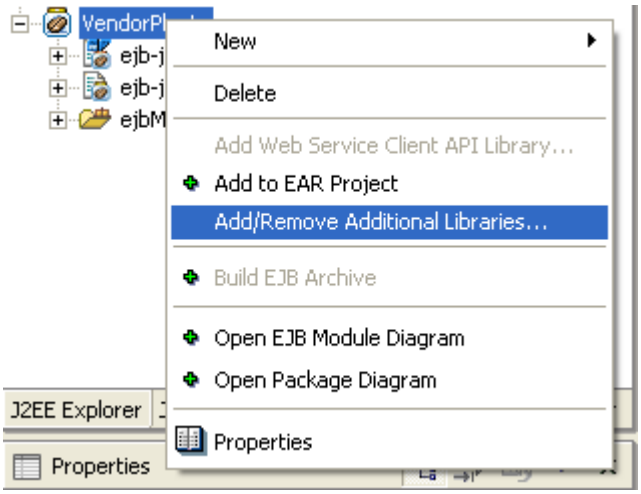


- You will find a Java Class is created in the EJB Module Project

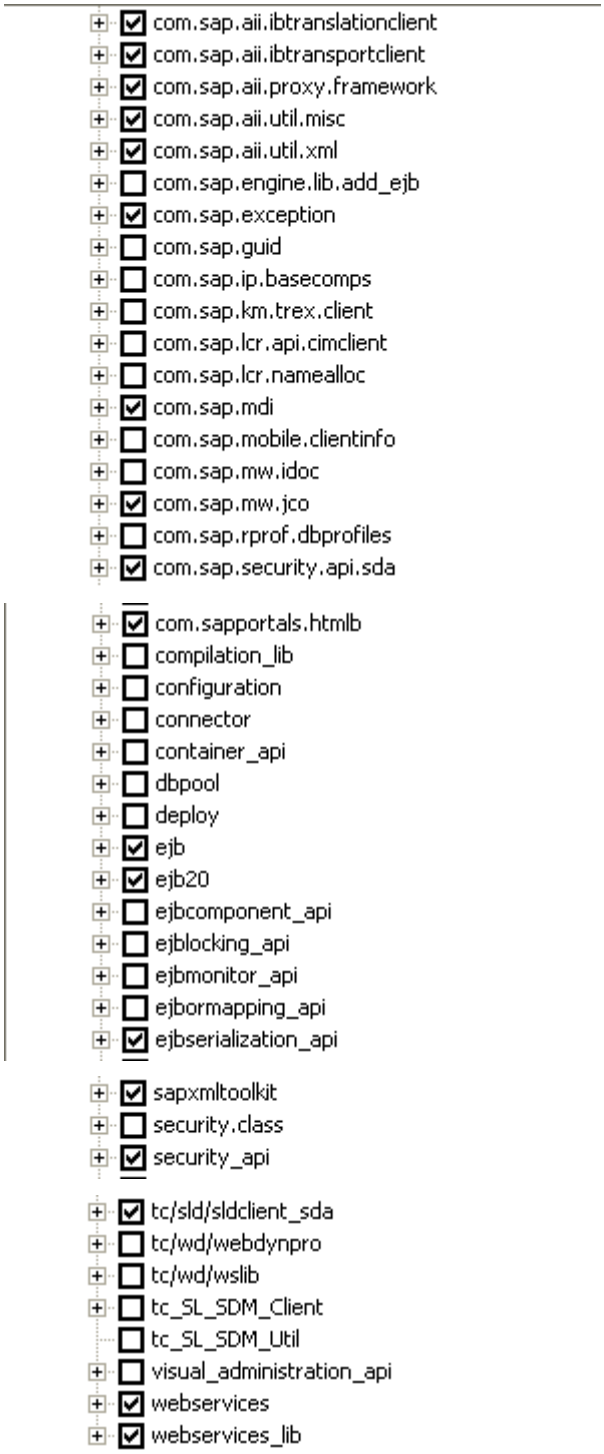
VendorPlants_PortType.java

```
package com.sap.vendorplants;
public class VendorPlants_PortType extends com.sap.aii.proxy.framework.core.AbstractProxy{

    private  static final com.sap.aii.proxy.framework.core.BaseProxyDescriptor staticDescriptor

    private  static final com.sap.aii.proxy.framework.core.GenerationInfo staticGenerationInfo

    public  VendorPlants_PortType () {
        super(staticDescriptor, staticGenerationInfo, com.sap.aii.proxy.framework.core.FactoryC

    }

    public com.sap.vendorplants.Zptp_Vendor_Get_Plants_Output zptp_Vendor_Get_Plants(com.sap.ve
        com.sap.aii.proxy.framework.core.BaseType $result = null;
        try {
        $result = send$(Zptp_Vendor_Get_Plants_Input, "urn:sap-com:document:sap:rfc:functions",
        }catch (com.sap.aii.proxy.framework.core.ApplicationFaultException e){
         throw createExceptionWrongExceptionType$(e);}
        if (($result == null) || ($result instanceof com.sap.vendorplants.Zptp_Vendor_Get_Plant
        return (com.sap.vendorplants.Zptp_Vendor_Get_Plants_Output)$result;
        } else { throw createExceptionWrongType$($result);}
    }

}
```

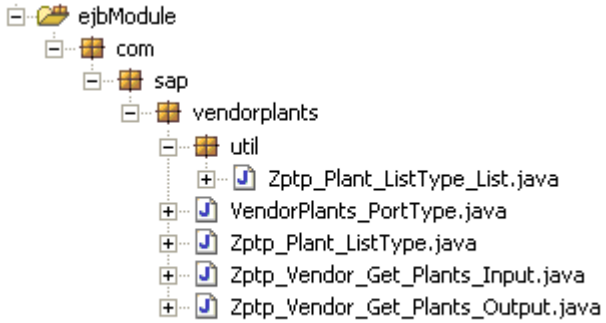
- You need to reference few jar files in the project. For that right click on Project and select Add/Remove Additional Libraries.



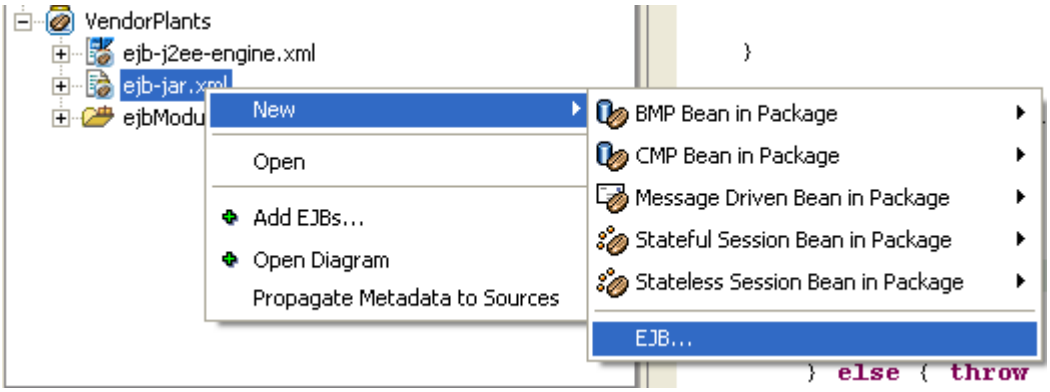
- Select Following Libraries (Note that same libraries need to be added in the EAR)



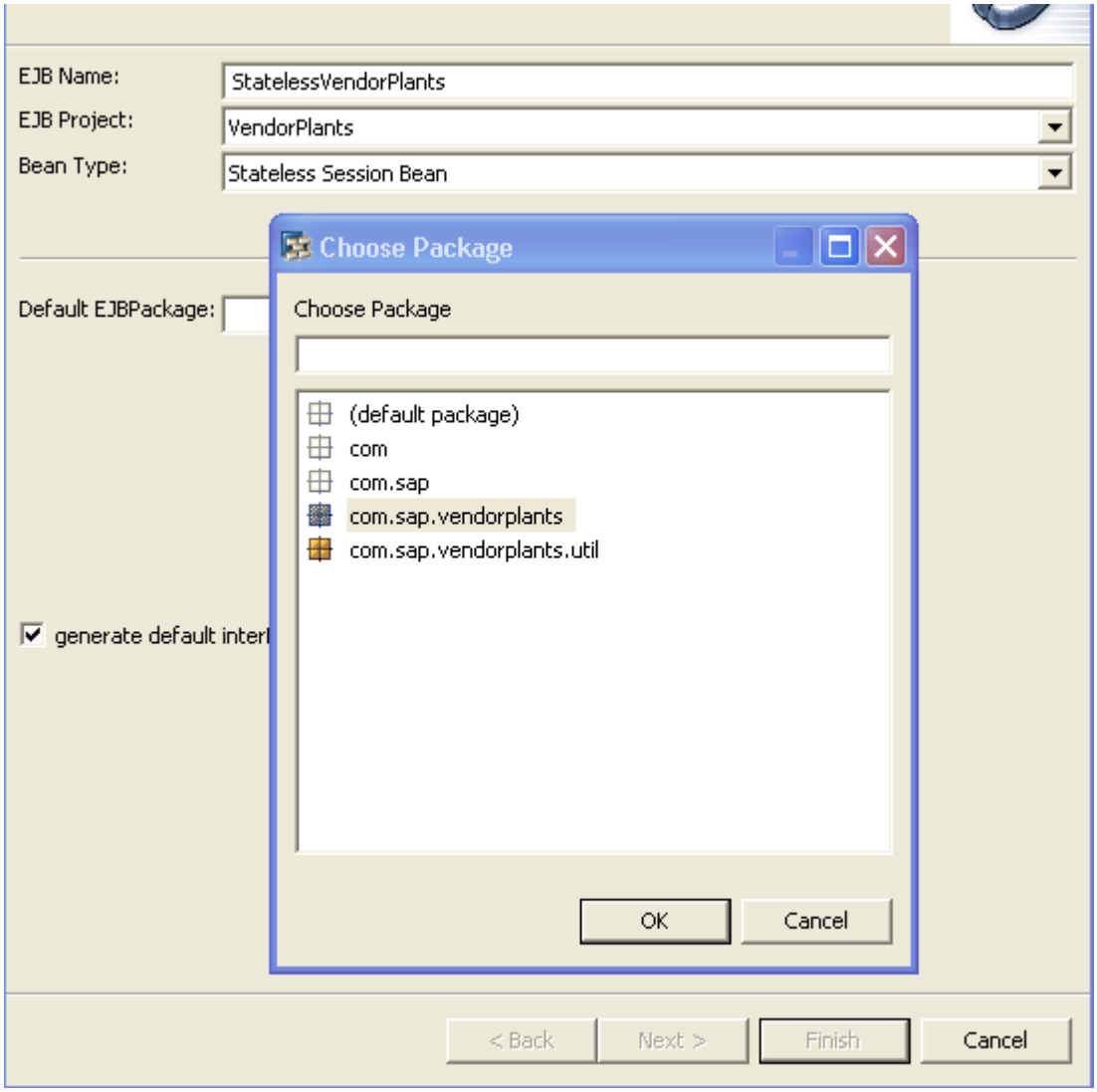
- Following Model classes will be created in the ejbmodule folder.



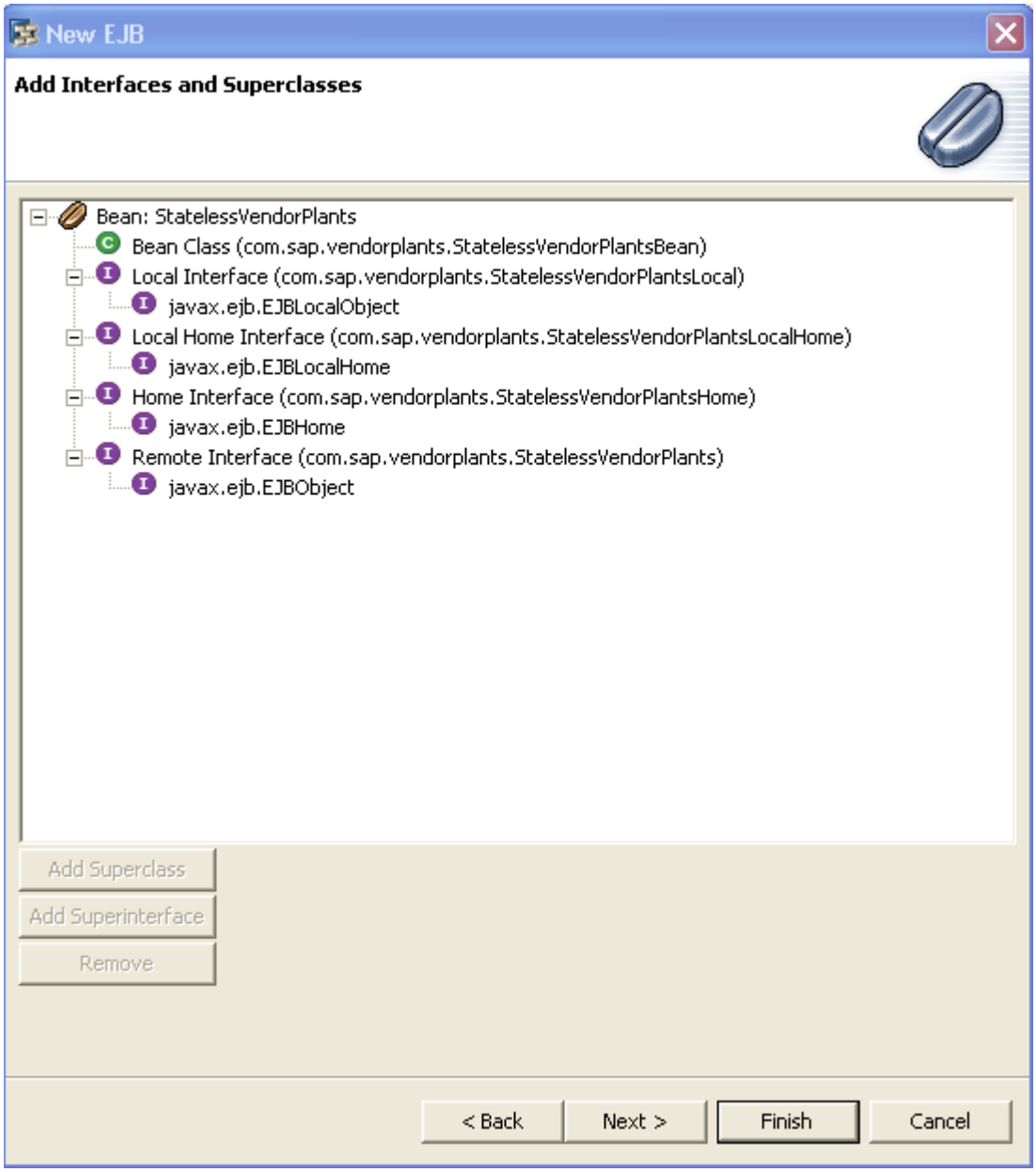
- Now, right click on the ejbjar.xml file and Select New Ejb.



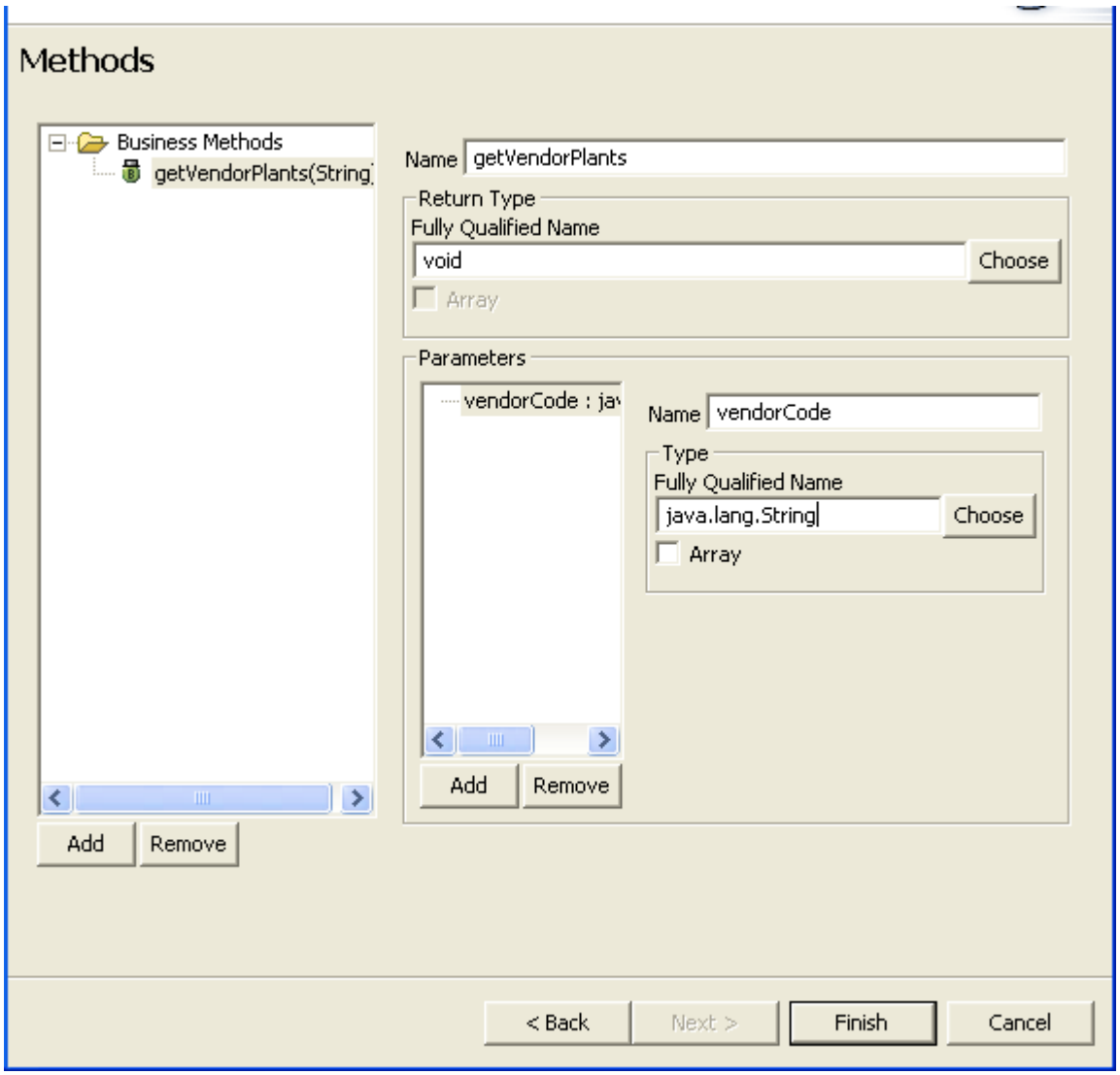
- Enter any EJB name and package name and select bean Type as Stateless Bean. Click on Ok.



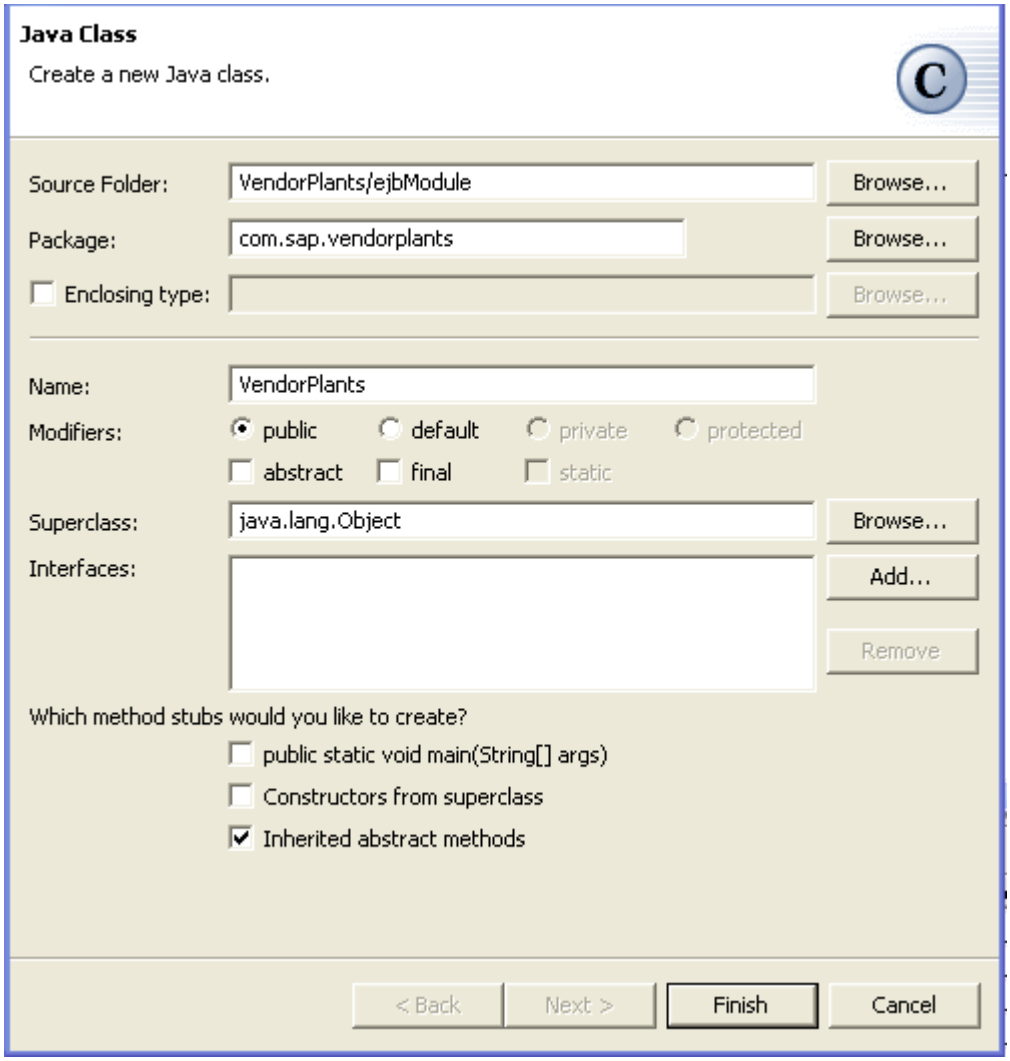
- Click on Next Button and you will find Bean classes and interfaces were created in the EJB Module Project.



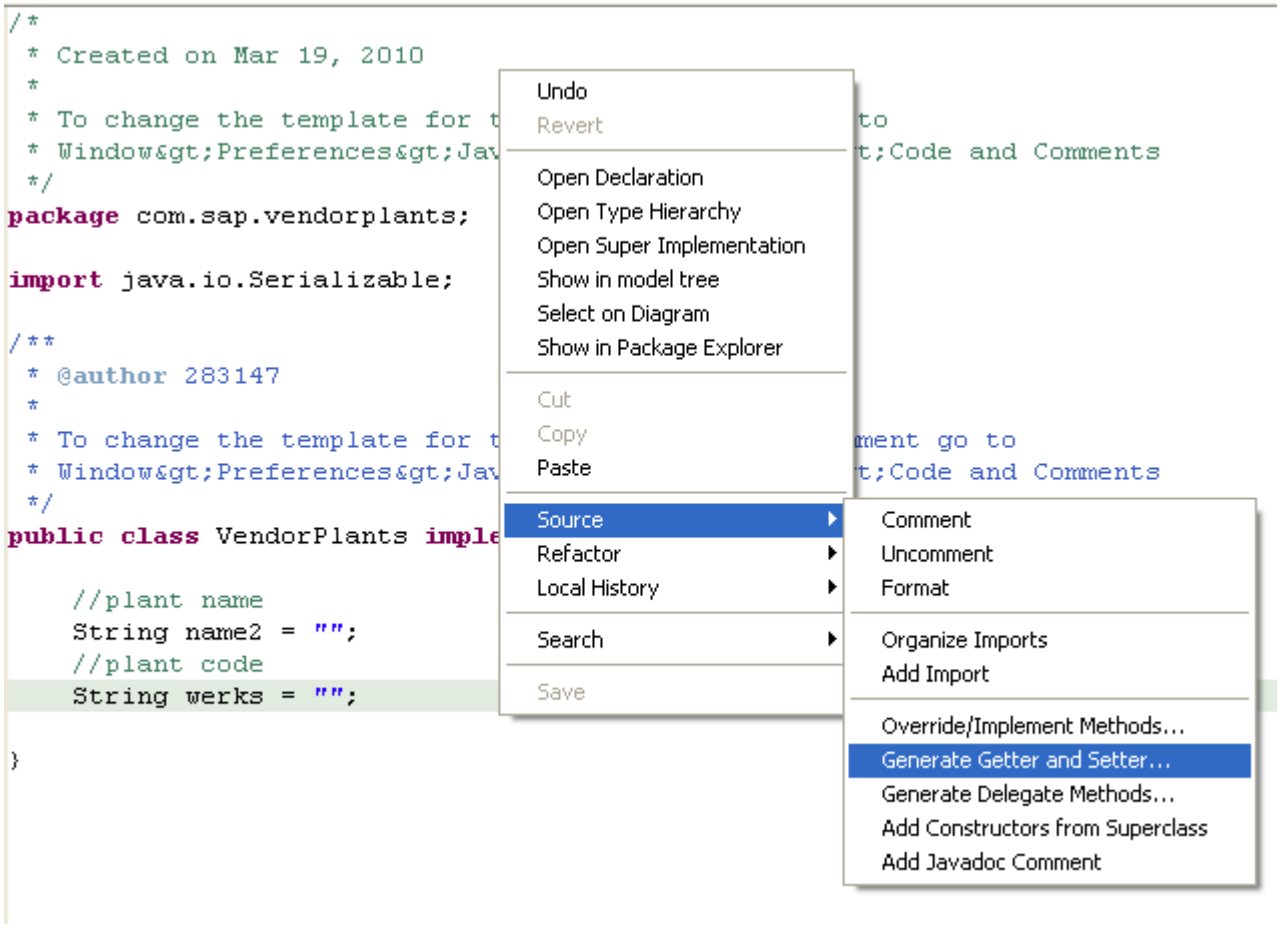
- Add business method getVendorPlants and a parameter (vendorCode) of type String and click on Finish (User has to pass VendorCode to the business method getVendorPlants of EJB).



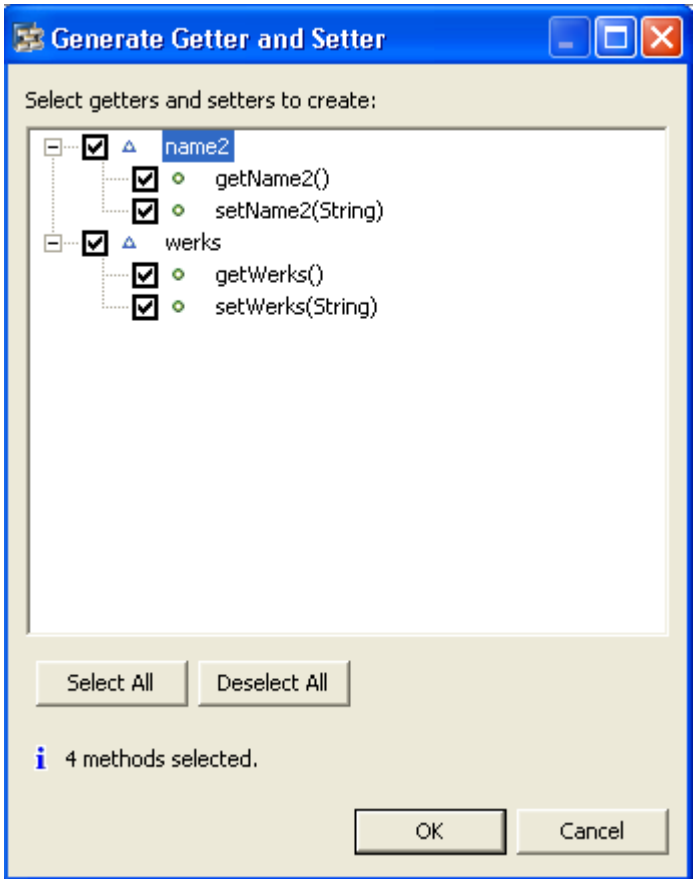
- We need to create an auxiliary class to store the data retrieved form the RFC. Right Click on ejbmodule folder and Select New Java Class. Enter class name and package name. In the same dialog box Select java.io.serializable interface. (Your auxiliary class should implement this interface)



- Open the class file and define two String variables name2 and werks. Right click and Select Source -> generate getter and setter methods.



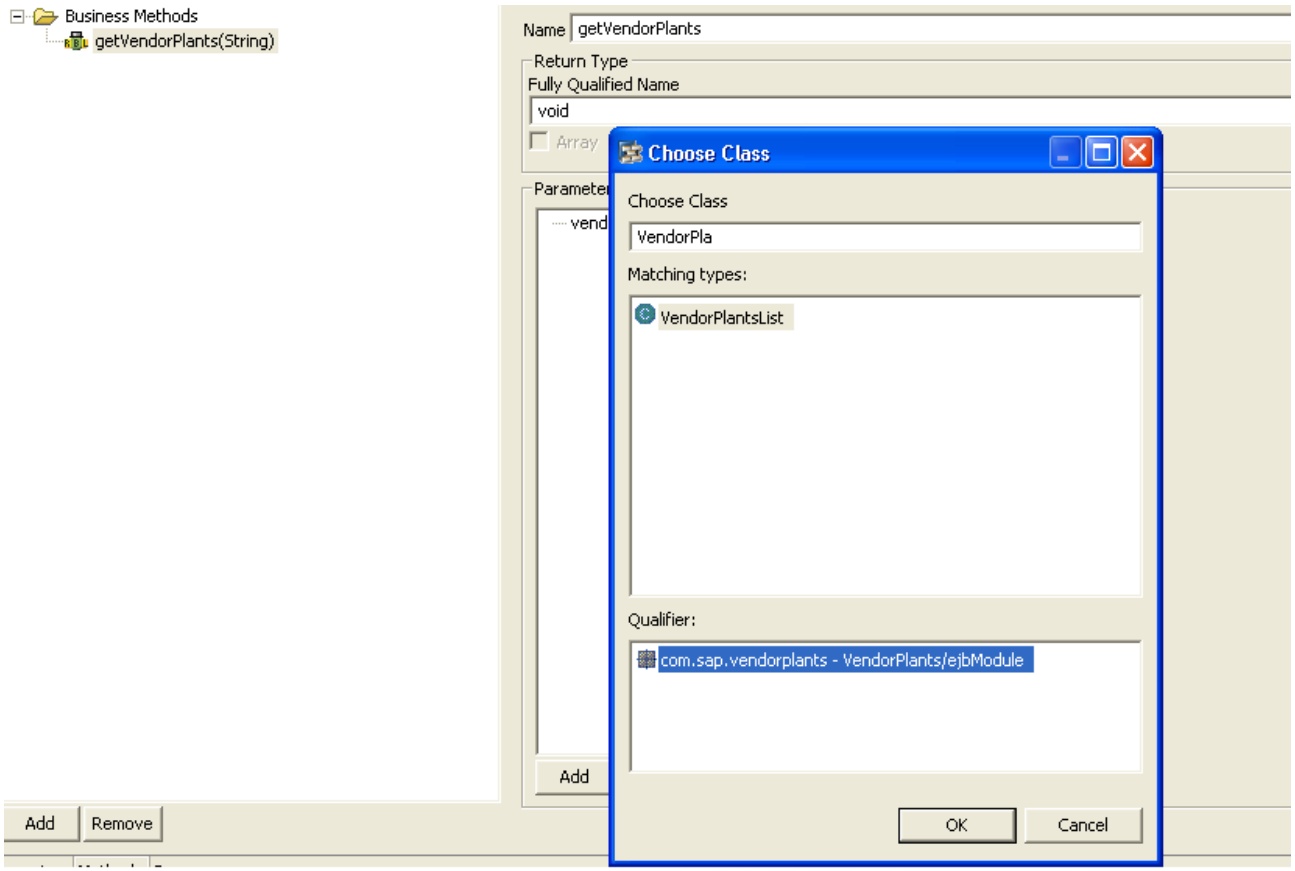
- Select all and click on Ok.



- Repeat the same procedure and create another auxiliary class VendorPlantsList. Define variables errorMessage as type String & vendorPlants as type VendorPlants[] (this is used to store the data and will be the return type of the ejb business method.

```
public class VendorPlantsList implements Serializable {

    //error message
    String errorMessage = "";
    //Vendor Plants List
    VendorPlants[] vendorPlants = null;

    /**
     * @return
     */
    public String getErrorMessage() {
        return errorMessage;
    }

    /**
     * @return
     */
    public VendorPlants[] getVendorPlants() {
        return vendorPlants;
    }

    /**
     * @param string
     */
    public void setErrorMessage(String string) {
        errorMessage = string;
    }

    /**
     * @param plantses
     */
    public void setVendorPlants(VendorPlants[] plantses) {
        vendorPlants = plantses;
    }

}
```

- Now, open EJB's business method getVendorPlants() and change the return type of the method to VendorPlantsList (select single dimensional array) and click on Ok.



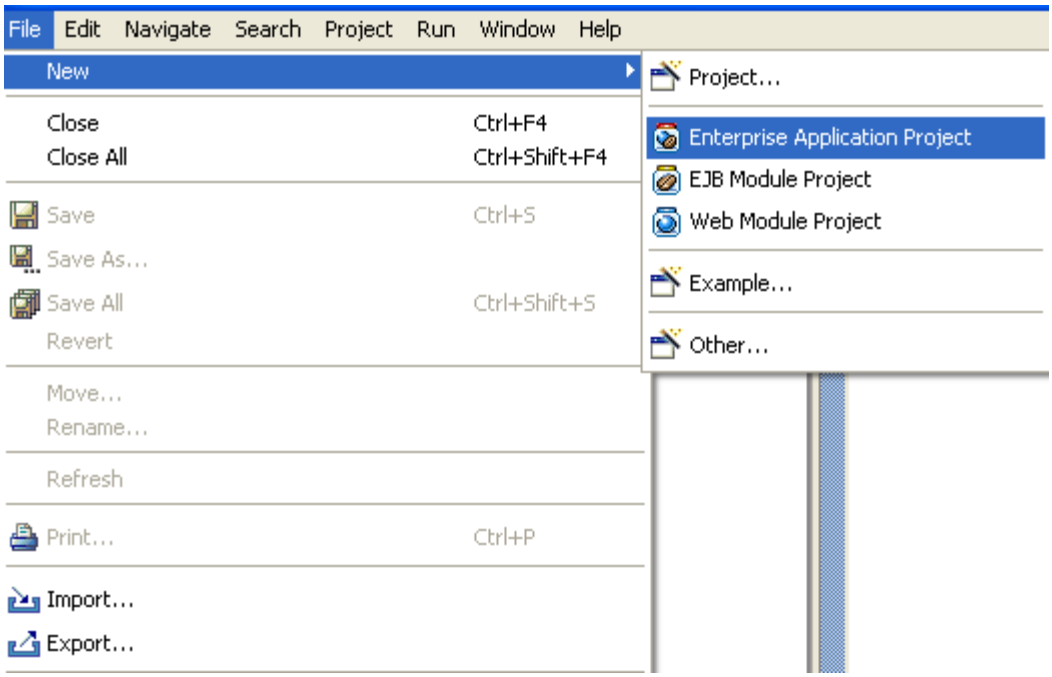
- Write following code in the business method.

```
public VendorPlantsList[] getVendorPlants(String vendorCode) {
            // TODO : Implement
            //creating jco client
            JCO.Client jcoclient = null;

            VendorPlantsList[] vendorPlantsList = null;
            VendorPlants [] vendorPlants = null;
            VendorPlantsList vendorPlantsDetails = new VendorPlantsList();

            try {
//using existing jco from portal "WD_MODELDATA_DEST" which is connected to
required backend SAP system
                  IWDJCOClientConnection client =
                  WDSystemLandscape.getJCOClientConnection("WD_MODELDATA_DEST");
                  jcoclient = client.getClient();
                  Properties props = jcoclient.getProperties();
                  jcoclient.connect();

                        if(jcoclient.isAlive())
                              {
                              //setting input parametres
```

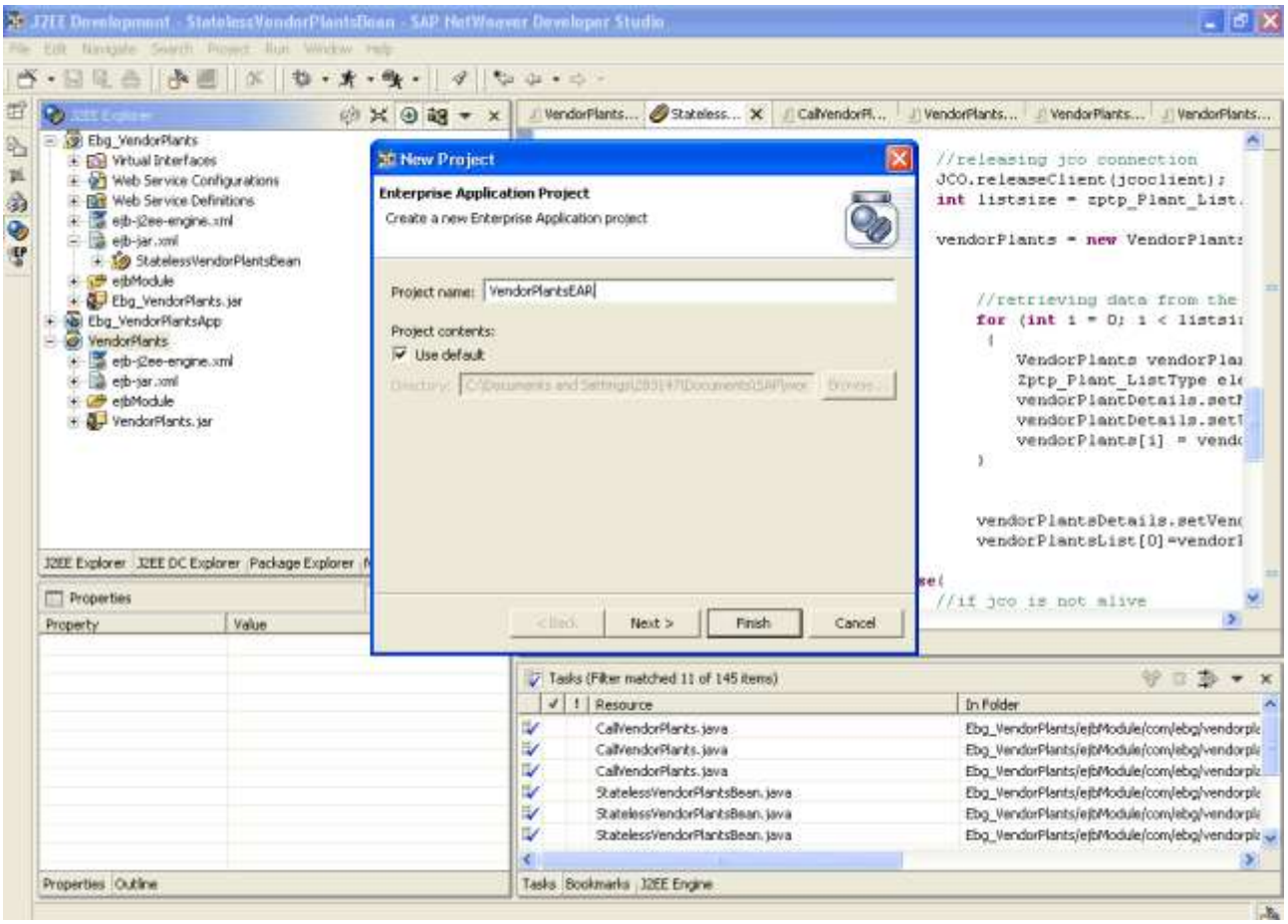
```
        Zptp_Vendor_Get_Plants_Output zptp_Vendor_Get_Plants_Output = null;
Zptp_Vendor_Get_Plants_Input zptp_Vendor_Get_Plants_Input = new
Zptp_Vendor_Get_Plants_Input();

        zptp_Vendor_Get_Plants_Input.setVendor_No(vendorCode);

        vendorPlantsList = new VendorPlantsList[1];

        Zptp_Plant_ListType_List zptp_Plant_List = new Zptp_Plant_ListType_List();

        VendorPlants_PortType vendorPlants_PortType = new VendorPlants_PortType();


        vendorPlants_PortType.messageSpecifier.setJcoClient(jcoclient);

        zptp_Vendor_Get_Plants_Output =
vendorPlants_PortType.zptp_Vendor_Get_Plants(zptp_Vendor_Get_Plants_Input);
        zptp_Plant_List = zptp_Vendor_Get_Plants_Output.get_as_listPlants();

        //releasing jco connection

        JCO.releaseClient(jcoclient);
        int listsize = zptp_Plant_List.size();
        vendorPlants = new VendorPlants[listsize];
//retrieving data from the list structure and storing it in array of type
VendorPlants[]
                    for (int i = 0; i < listsize; i++)
                            {
VendorPlants vendorPlantDetails = new VendorPlants();
Zptp_Plant_ListType elementPlantListType =
zptp_Plant_List.getZptp_Plant_ListType(i);


        vendorPlantDetails.setName2(elementPlantListType.getName2());

        vendorPlantDetails.setWerks(elementPlantListType.getWerks());

        vendorPlants[i] = vendorPlantDetails;

                            }


        vendorPlantsDetails.setVendorPlants(vendorPlants);


        vendorPlantsList[0]=vendorPlantsDetails;
              }
                    else{
                    //if jco is not alive
        vendorPlantsList = new VendorPlantsList[1];

        vendorPlantsDetails.setErrorMessage("Connection could not be established");

              vendorPlantsList[0]=vendorPlantsDetails;
```

```
                                                                    }

                                                                }
                catch (WDSystemLandscapeException e) {
                // if jco of perticular name not found
vendorPlantsList = new VendorPlantsList[1];

vendorPlantsDetails.setErrorMessage(e.getMessage());

vendorPlantsList[0]=vendorPlantsDetails;
}
catch(ApplicationFaultException afx)
{
vendorPlantsList = new VendorPlantsList[1];

vendorPlantsDetails.setErrorMessage(afx.getMessage());

vendorPlantsList[0]=vendorPlantsDetails;
}
catch(SystemFaultException sfx)
{
vendorPlantsList = new VendorPlantsList[1];

vendorPlantsDetails.setErrorMessage(sfx.getMessage());

vendorPlantsList[0]=vendorPlantsDetails;
}

return vendorPlantsList;

}
```

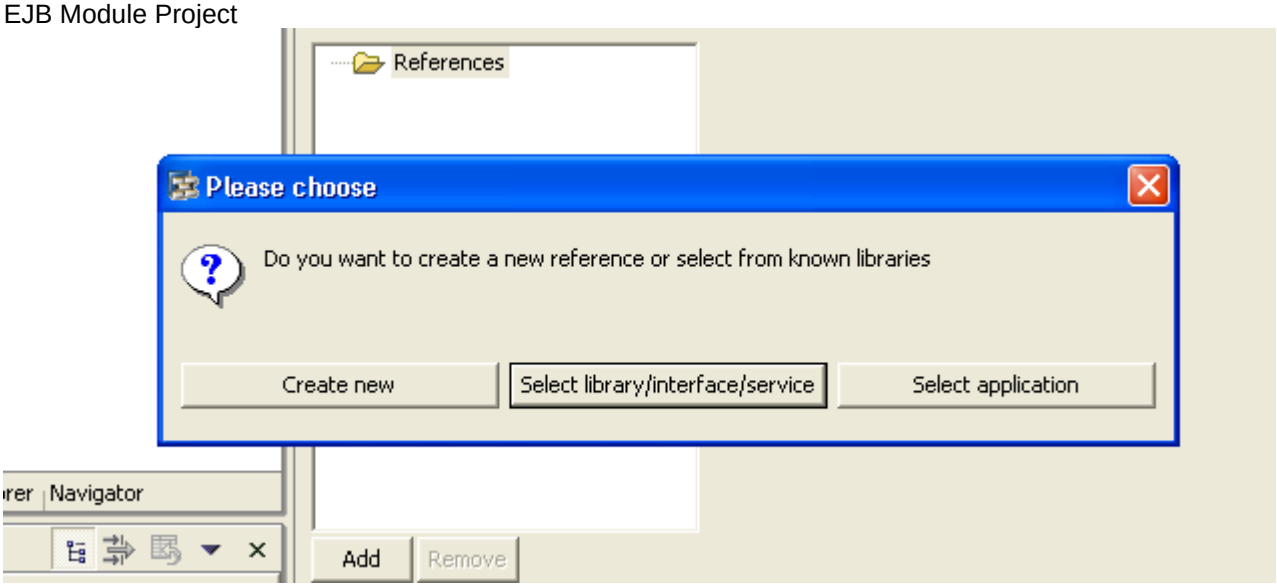
- Right Click on the Project and Build.
- Now, Select File -> New -> Enterprise application Project (This is required to deploy the ejb on the WEB AS)



- Enter the name for Application Project and click on Next Button



- Open application-j2ee-engine.xml and click on references and Click on Select Library/Interface/service button to add all the additional Libraries which you have selected for the EJB Module Project



- Right click on Enterprise Application Project and click on Build application Archive.
- Deploy the .ear file on WEB AS.

Exposing EJB as a Web Service

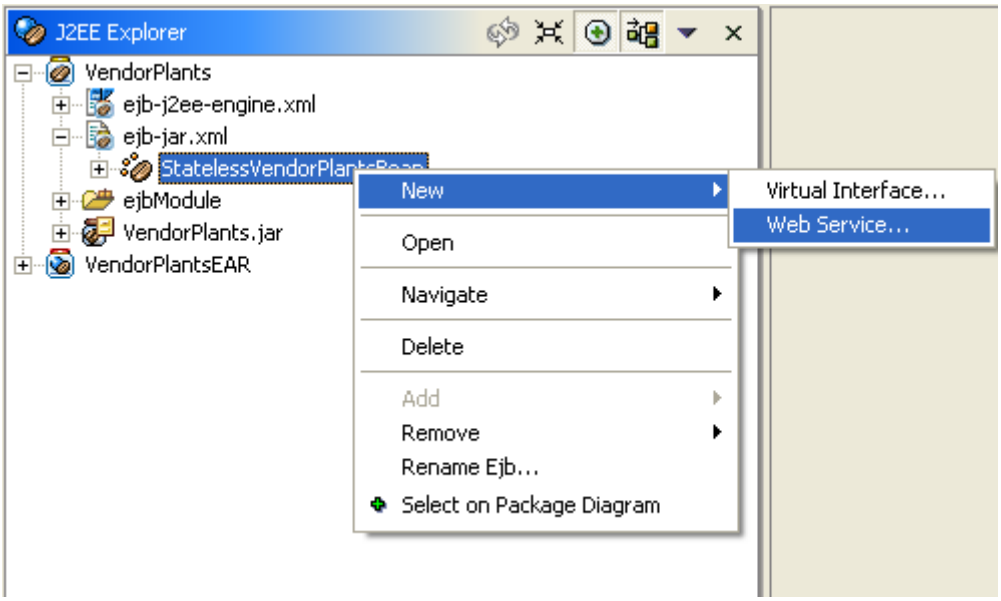
Supported types

Supported Types in WS Endpoints

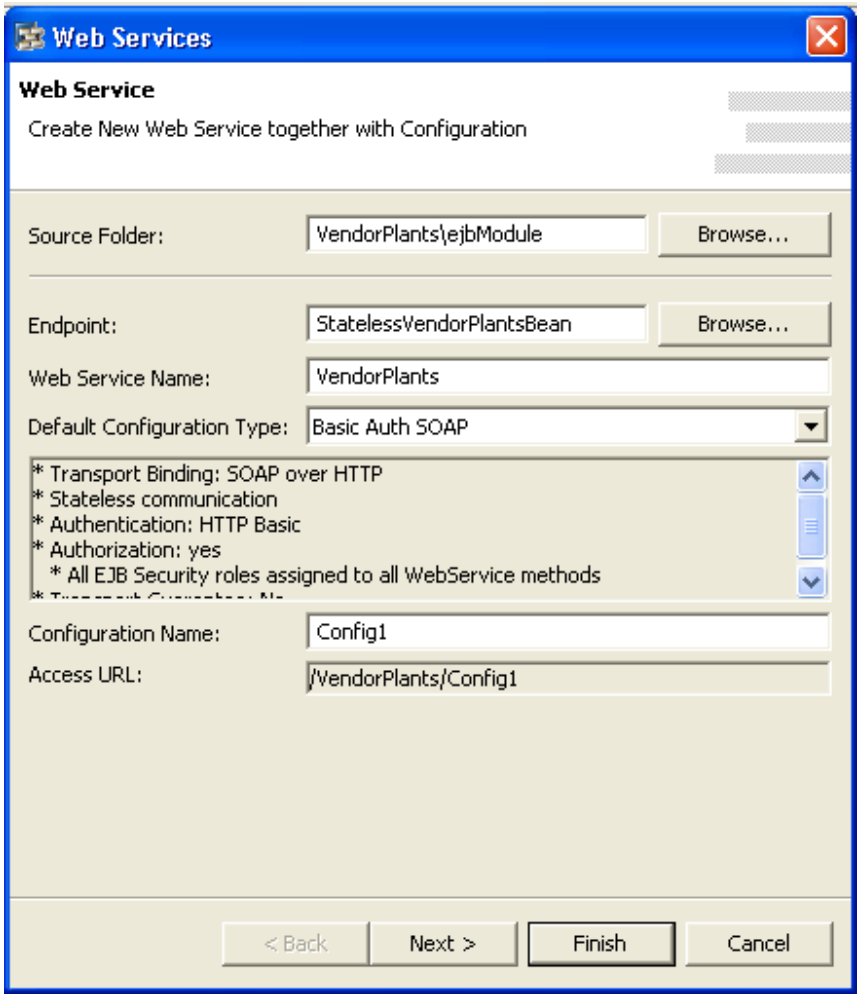
The following types from java*.packages are supported in endpoints of Web Services:

Simple Types Wrappers	java.lang.Void java.lang.Boolean java.lang.Byte java.lang.Short java.lang.Integer java.lang.Long java.lang.Float java.lang.Double java.lang.String
Marker Interfaces	java.io.Serializable java.lang.Cloneable java.lang.Comparable
Array classes	java.util.ArrayList java.util.HashSet java.util.LinkedList java.util.List java.util.Stack java.util.Vector
Date/Time	java.util.Calendar java.util.Date java.util.GregorianCalendar java.sql.Date java.sql.Time
Long numbers	java.math.BigInteger java.math.BigDecimal
Exception	java.lang.Throwable java.lang.Exception java.rmi.RemoteException
Object	java.lang.Object[3]

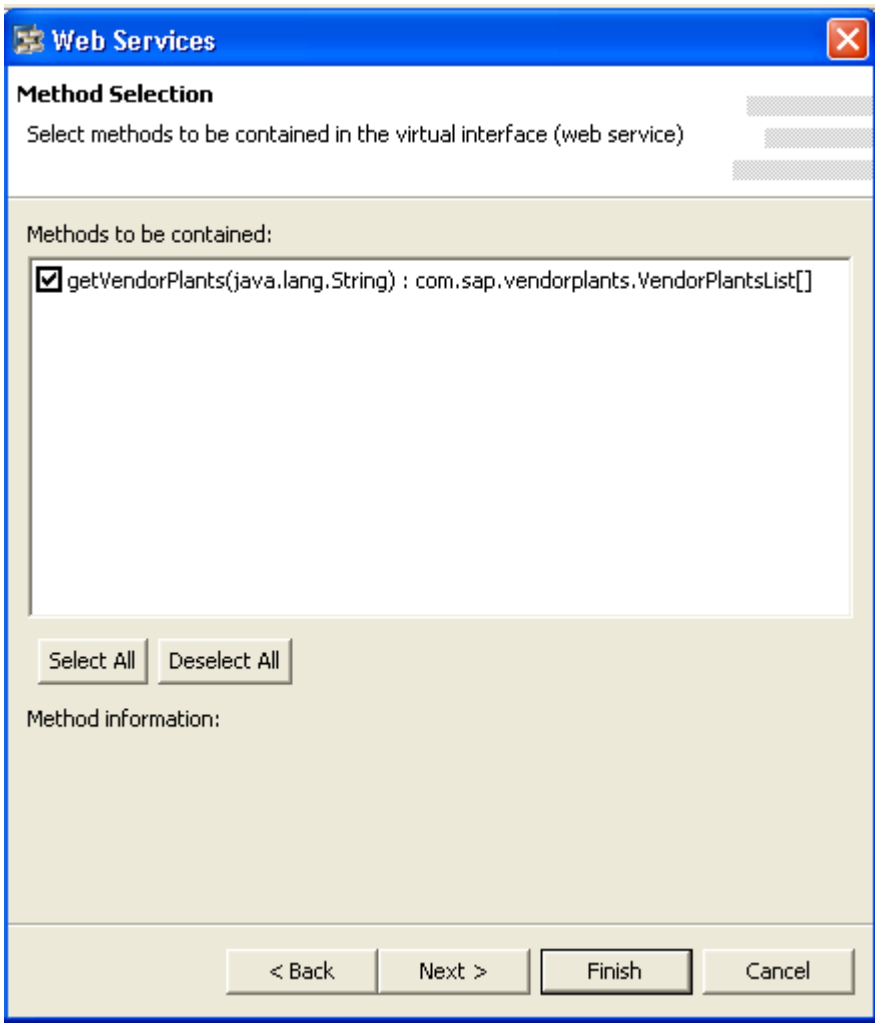
- Right click on EJB and Select New -> Web Service.



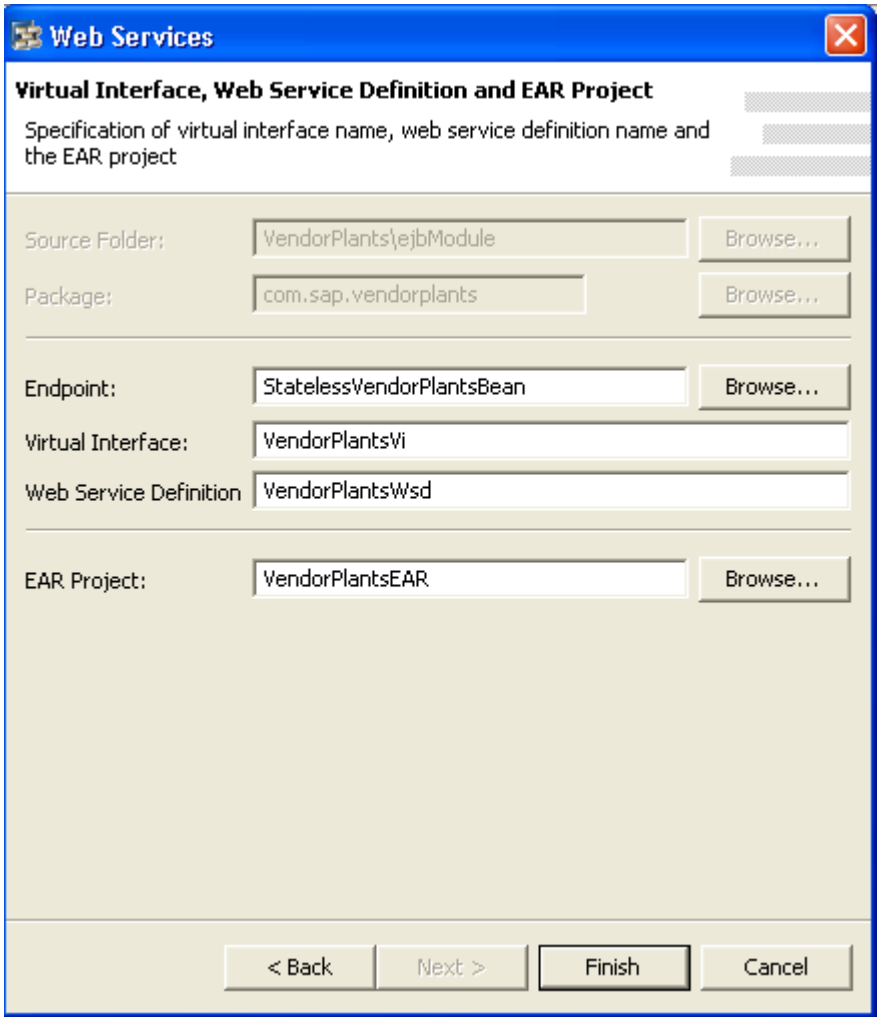
- Enter name of the Web Service and select type Simple SOAP / Basic Auth SOAP
- Note that Basic Auth SOAP will require the Username and password to execute Web Service.



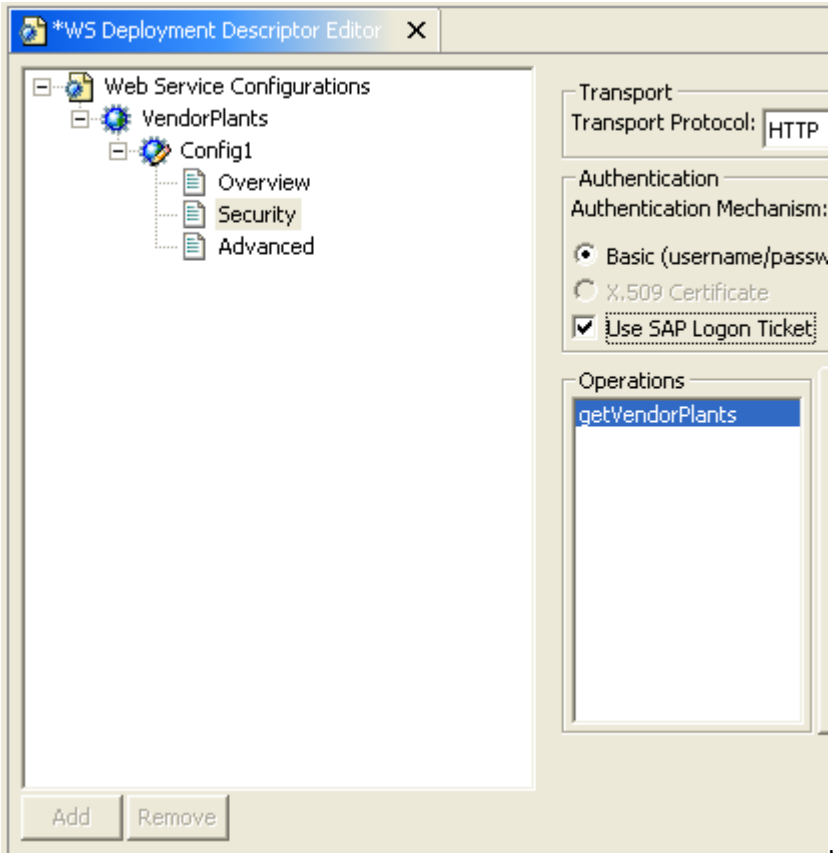
- Click on Next button.
- Select EJB methods you want to expose and click on Next Button.



- Select Endpoint Bean and EAR project and click on finish Button.

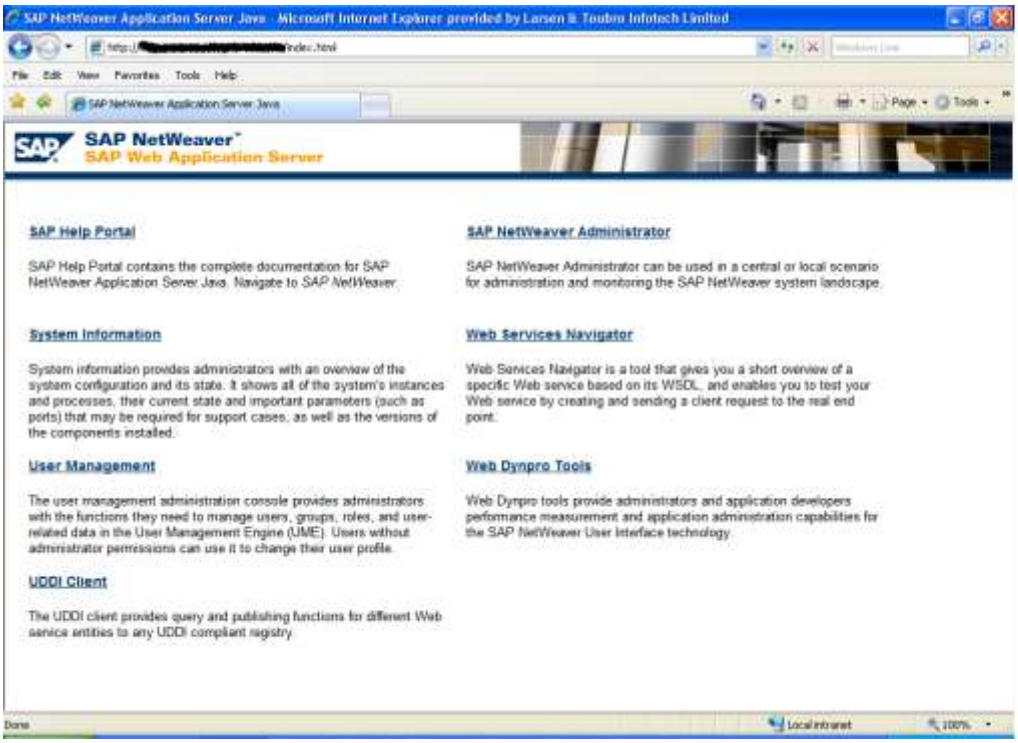


- Open Web Service configuration and Select Security. Check the option get SAP Logon Ticket. So that the SAP Logon ticket can be used for the entire user session.



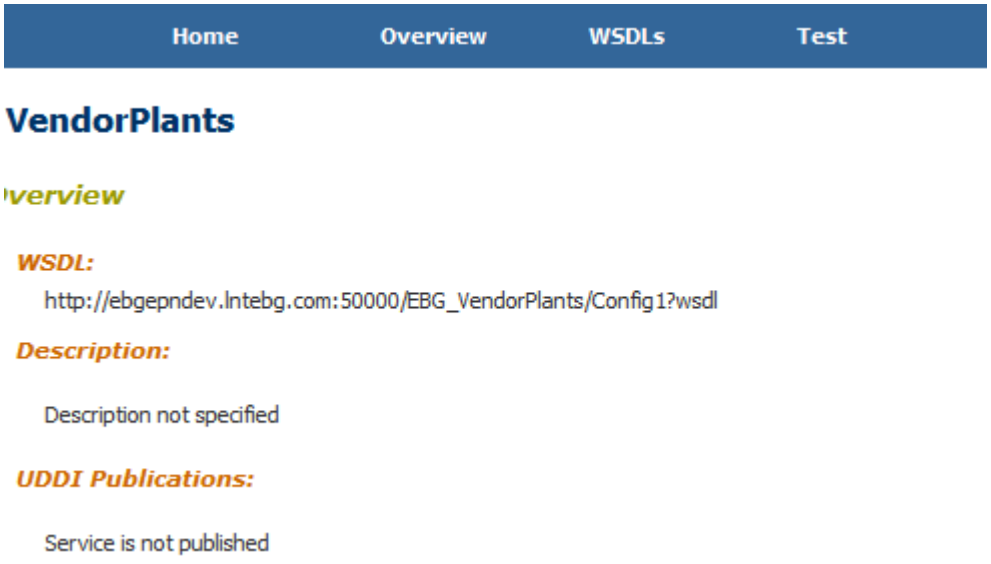
- Save the changes and build both the projects and deploy the EAR file on the WEB AS.
- Open the index.html page of the WEB AS

- Click on Web Service Navigator option (Note that use should have administrative rights to run the web services through navigator)

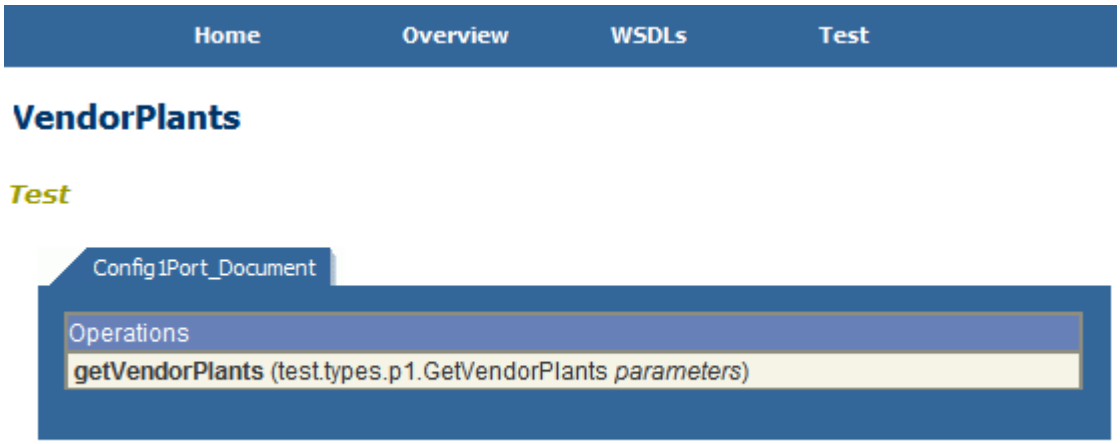


- Click on Web Service which you have created.

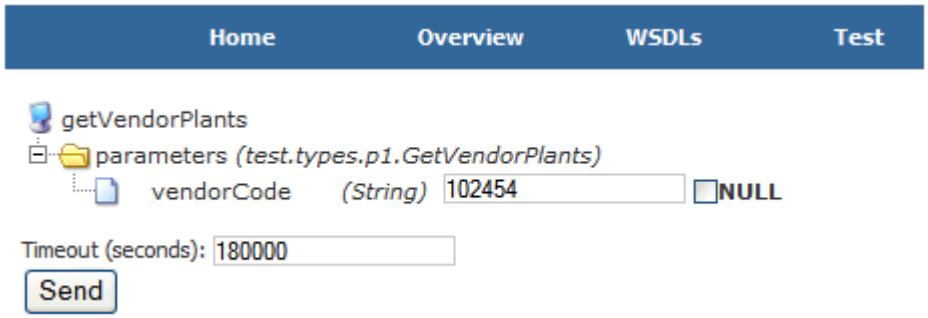
- Click on Test option



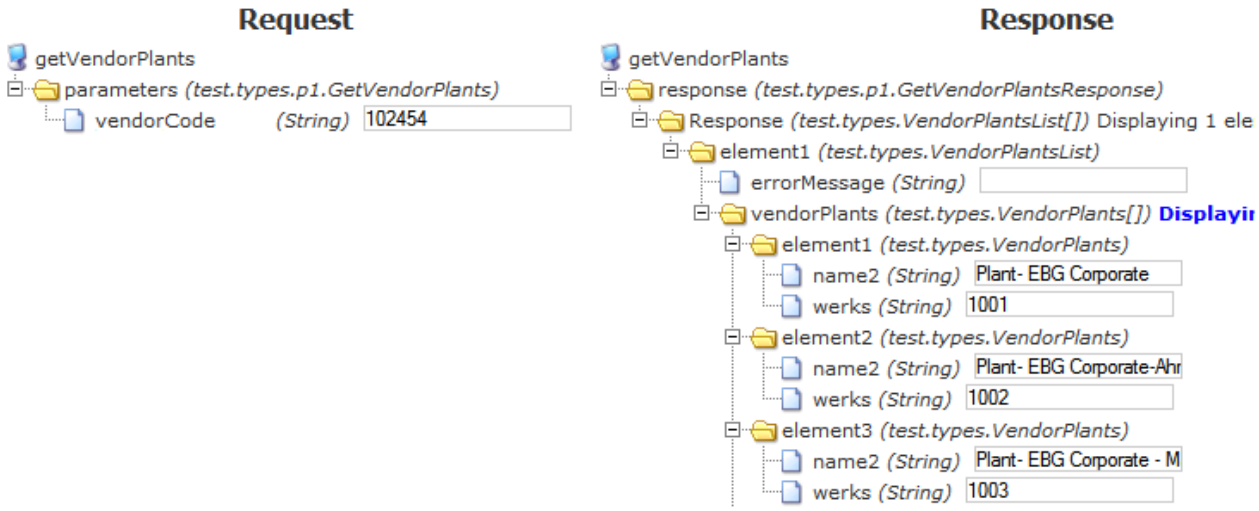
- Click on the Web Service Operation which you have created.



- Click on the Web Service Operation which you have created. You will find the input parameter of your EJB here.



- Enter valid Vendor Code (which exists in the backend SAP System) and click on Send button.
- If your Web Service is Basic Auth SOAP type, then WEB AS will prompt you to enter Username and password.
- The Result will be displayed based on the Vendor Code you have entered.



Summary

Thus, we have understood the need of EJB, Web Service and procedure to create an EJB and expose it as a Web Service.

Related Content

http://help.sap.com/saphelp_nw70/helpdata/en/35/42e13d82fcfb34e10000000a114084/frameset.htm

http://help.sap.com/saphelp_nw70/helpdata/en/f7/af60f2e04d0848888675a800623a81/frameset.htm

http://www.sdn.sap.com/irj/scn/weblogs?blog=/pub/wlg/10300

http://help.sap.com/saphelp_nw70/helpdata/en/9b/dad1ae3908ee44a5caf57e10918be9/frameset.htm

http://help.sap.com/saphelp_nw70/helpdata/en/35/42e13d82fcfb34e10000000a114084/frameset.htm

Disclaimer and Liability Notice

This document may discuss sample coding or other information that does not include SAP official interfaces and therefore is not supported by SAP. Changes made based on this information are not supported and can be overwritten during an upgrade.

SAP will not be held liable for any damages caused by using or misusing the information, code or methods suggested in this document, and anyone using these methods does so at his/her own risk.

SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of this technical article or code sample, including any liability resulting from incompatibility between the content within this document and the materials and services offered by SAP. You agree that you will not hold, or seek to hold, SAP responsible or liable with respect to the content of this document.

References

[1] Developing an EJB Application, SAP AG
http://help.sap.com/saphelp_nw70/helpdata/en/b5/3f533e5ff4d064e10000000a114084/frameset.htm

[2] Web Services, SAP AG
http://help.sap.com/saphelp_nw70/helpdata/en/9b/dad1ae3908ee44a5caf57e10918be9/frameset.htm

[3] Restrictions for WS Endpoints, SAP AG
http://help.sap.com/saphelp_nw70/helpdata/en/f7/af60f2e04d0848888675a800623a81/frameset.htm